Лабораторная работа № 7
АСИММЕТРИЧНЫЕ АЛГОРИТМЫ В .NET

Цель работы

Изучить возможности реализации асимметричной криптосистемы средствами .NET

Методические указания

1. Класс AsymmetricAlgorithm.

Класс AsymmetricAlgorithm пространства имен System.Security.Cryptography среды .NET Framework представляет абстрактный базовый класс, от которого должны наследоваться все реализации алгоритмов асимметричного шифрования.

На схеме представлена иерархия наследования криптографических классов асимметричных алгоритмов шифрования.

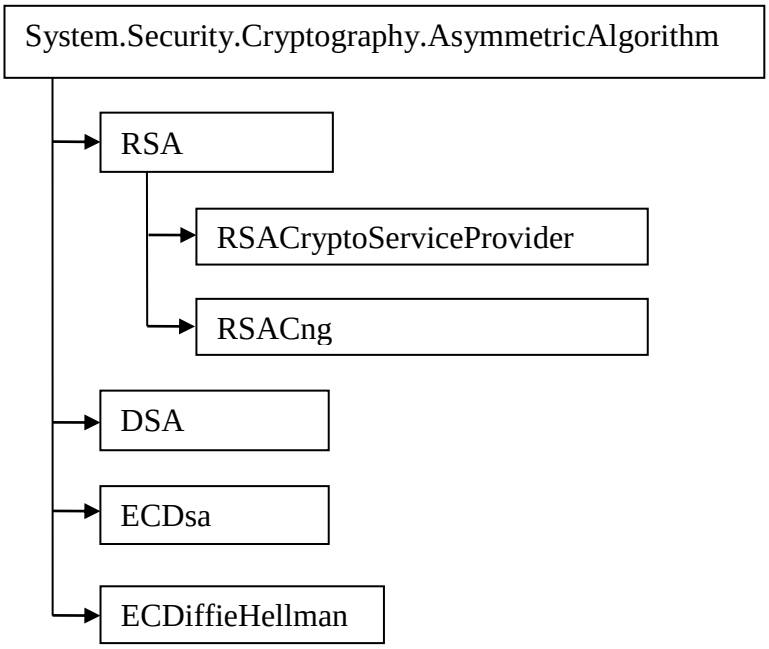

Рис. 1 – Иерархия наследования криптографических классов асимметричных алгоритмов шифрования

Класс AsymmetricAlgorithm предоставляет несколько общих полей, свойств и методов, которые можно использовать во всех асимметричных алгоритмах.

Член класса (поле, свойство или метод	Описание
KeySize и KeySizeValue	Получает или задает размер модуля ключа, используемого алгоритмом асимметричного шифрования. KeySize – это открытое свойство, KeySizeValue – защищенное поле, оба свойства возвращают целое значение, представляющее длину ключа в битах.
LegalKeySizes и LegalKeySizesValue	Указывает допустимые размеры ключей для текущего асимметричного алгоритма. LegalKeySizes – это открытое свойство, LegalKeySizesValue – защищенное поле, оба свойства возвращают массив KeySize.
KeyExchangeAlgorithm	Свойство при переопределении в производном классе возвращает имя алгоритма обмена ключами. В противном

	случае создается исключение NotImplementedException
SignatureAlgorithm	Свойство при реализации в производном классе возвращает имя алгоритма подписи. В противном случае создается исключение NotImplementedException
FromXmlString(String)	Метод, если переопределен в производном классе, восстанавливает объект AsymmetricAlgorithm из XML-строки. В противном случае создается исключение NotImplementedException
ToXmlString(Boolean)	Метод, если переопределен в производном классе, создает и возвращает представление текущего объекта AsymmetricAlgorithm в виде XML-строки. В противном случае создается исключение NotImplementedException

Класс RSA является производным от класса типа алгоритма AsymmetricAlgorithm. Это абстрактный класс, от которого наследуются конкретные классы, реализующие алгоритм RSA. Класс RSA позволяет иметь несколько реализаций алгоритма RSA. В настоящее время класс RSA наследуется двумя классами: RSACryptoServiceProvider и RSACng.

Класс RSACryptoServiceProvider представляет собой оболочку для реализации алгоритма RSA в CryptoAPI, а класс RSACng написан с использованием криптографии следующего поколения (CNG) - это новейшая реализация, дальнейшая разработка которой продолжается. Алгоритмы CNG доступны в Windows Vista и последующих версиях.

2. Класс RSACryptoServiceProvider

Как было сказано выше, класс RSACryptoServiceProvider выполняет асимметричное шифрование и расшифрование с помощью реализации алгоритма RSA, предоставляемой криптопровайдером CSP (Cryptographic Service Provider). Этот класс не наследуется.

Конструкторы класса RSACryptoServiceProvider представлены в таблице:

Имя	Описание конструктора
RSACryptoServiceProvider()	Инициализирует новый экземпляр класса RSACryptoServiceProvider, используя ключ по умолчанию.
RSACryptoServiceProvider(CspParameters)	Инициализирует новый экземпляр класса RSACryptoServiceProvider с заданными параметрами.
RSACryptoServiceProvider(Int32)	Инициализирует новый экземпляр класса RSACryptoServiceProvider с указанным размером ключа.
RSACryptoServiceProvider(Int32, CspParameters)	Инициализирует новый экземпляр класса RSACryptoServiceProvider указанным размером ключа и параметрами.

Некоторые свойства класса RSACryptoServiceProvider представлены в таблице:

Имя свойства	Описание свойства
KeySize	Возвращает размер текущего ключа. (Наследуется от AsymmetricAlgorithm.KeySize.)
LegalKeySizes	Возвращает размеры ключа, которые поддерживаются алгоритмом асимметричного шифрования. (Наследуется от AsymmetricAlgorithm.)
PersistKeyInCsp	Возвращает или задает значение, указывающее, следует ли сохранить ключ поставщике служб

	шифрования CSP (т.е. в экземпляре класса CspParameters).
UseMachineKeyStore	Получает или задает значение, указывающее, следует ли сохранять ключ в хранилище ключей компьютера, а не в хранилище профилей пользователей.

Некоторые методы класса RSACryptoServiceProvider представлены в таблице:

Имя метода	Описание метода
Clear()	Освобождает все ресурсы, используемые классом AsymmetricAlgorithm. (Наследуется от AsymmetricAlgorithm.)
Dispose()	Освобождает все ресурсы, используемые текущим экземпляром класса AsymmetricAlgorithm. (Наследуется от AsymmetricAlgorithm.)
Encrypt(byte[] rgb, bool fOAEP)	Шифрует данные с помощью алгоритма RSA.
Decrypt(byte[] rgb, bool fOAEP)	Расшифровывает данные с помощью алгоритма RSA.
ToString()	Возвращает строковое представление текущего объекта. (Наследуется от Object.)
ExportParameters(Boolean)	Экспортирует параметры алгоритма RSA в структуру RSAParameters. (Переопределяет RSA.ExportParameters(Boolean).)
ImportParameters(RSAParameters)	Импортирует параметры алгоритма RSA из структуры RSAParameters в объект класса RSACryptoServiceProvider. (Переопределяет RSA.ImportParameters(RSAParameters).)
ToXmlString(Boolean)	Создает и возвращает строку XML, содержащую ключ текущего объекта RSA.(Наследуется от RSA.)
FromXmlString(String)	Инициализирует объект RSA, используя данные ключа из строки XML.(Наследуется от RSA.)

Основные методы класса RSACryptoServiceProvider – Encrypt и Decrypt.

Метод Encrypt – шифрует данные с помощью алгоритма RSA.

Синтаксис метода: ***public byte[] Encrypt(byte[] rgb, bool fOAEP)***

Первый параметр ***rgb*** - данные, предназначенные для шифрования (массив байт).

Максимально допустимая длина параметра ***rgb*** определяется длиной блока алгоритма RSA. Длина блока в свою очередь зависит от используемой версии ОС Microsoft Windows и метода заполнения блока. При установке пакета High Encryption – 16 байтов, иначе – 5 байтов. (Более подробную информацию можно получить из описания метода encrypt в документации MSDN).

Второй параметр ***fOAEP*** определяет метод дополнения блока для алгоритма RSA.

Дополнение блока необходимо потому, что алгоритм RSA требует фиксированного размера блока, а размер входных данных, в общем случае, может не соответствовать этому размеру.

Для параметра *fOAEP* рекомендуется использовать значение ***true***, при этом для дополнения блока будет применяться технология OAEP.

OAEP (Optimal Asymmetric Encryption Padding) – оптимальное дополнение для асимметричного шифрования. Это техника дополнения PKCS#1 v2, разработанная специально для алгоритма RSA, которая обеспечивает дополнение, гораздо более качественное с точки зрения безопасности, в сравнении с техникой PKCS#1 v1.5.

Дополнение доступно на компьютерах под управлением ОС Microsoft Windows XP и всех последующих версий при установке соответствующего пакета шифрования.

Значение ***false*** - для дополнения блока будет использоваться метод PKCS#1 v1.5.

Возвращаемое значение: метод возвращает зашифрованный массив байтов.

Исключения:

CryptographicException – не удалось получить криптопровайдера (CSP), либо длина параметра ***rgb*** превышает максимально допустимую длину, либо параметр ***fOAEP*** имеет значение true, а заполнение OAEP не поддерживается.

ArgumentNullException – параметр ***rgb*** имеет значение ***null***.

Метод Decrypt - расшифровывает данные с помощью алгоритма RSA.

Синтаксис метода аналогичен синтаксису метода *Encrypt*:

public byte[] Decrypt(byte[] rgb, bool fOAEP)

Пример кода шифрования/расшифрования данных с использованием класса RSACryptoServiceProvider и методов Encrypt/Decrypt можно найти в MSDN.

Для задания нужного размера ключа можно использовать специальный конструктор класса RSACryptoServiceProvider.

3. Ключи асимметричного шифрования. Хранение ключей.

В асимметричных алгоритмах шифрования используются пары ключей. Если шифрование выполнялось одним ключом из пары, то расшифрование производится другим. Открытые (public) ключи могут передаваться другим лицам для проверки цифровых подписей и шифрования пересылаемых данных.

Личные (private) ключи не могут быть экспортированы; они используются для создания цифровых подписей и расшифрования данных. Личный ключ должен быть известен только его владельцу.

За хранение и разрушение ключей отвечает криптопровайдер. Программист не имеет доступа непосредственно к двоичным данным ключа, за исключением операций экспорта открытых ключей.

Криптопровайдер поддерживает защищенные области, называемые **контейнерами ключей**. Контейнеры позволяют приложениям сохранять и использовать в дальнейшем сгенерированные один раз ключи, обеспечивая защиту самого ключа от злоумышленника.

Контейнеры бывают двух типов – **пользовательские** (этот тип используется по умолчанию) и **машинные** (CRYPT_MACHINE_KEYSET). Пользовательский контейнер доступен только приложениям, выполняемым от имени владельца контейнера. Приложение может использовать такой контейнер для сохранения персональных ключей данного пользователя. Доступ к машинным контейнерам разрешен только администраторам. В них обычно сохраняются ключи, используемые сервисами и системными программами.

Свойство UseMachineKeyStore класса RSACryptoServiceProvider получает или задает значение, указывающее, следует ли сохранять ключи в контейнере ключей компьютера, либо в контейнере профилей пользователей. По умолчанию ключи сохраняются в пользовательском контейнере.

В модели криптографии .NET Framework ключи для шифрования/расшифрования и создания/проверки подписей разделены. Называются они соответственно «пара для обмена ключами» и «пара для подписи».

Т. о. база данных ключей состоит из контейнеров, в каждом из которых хранятся ключи, принадлежащие определенному пользователю. Контейнер ключей имеет уникальное имя. В контейнере может существовать не более одной пары ключей подписи, одной пары ключей обмена и одного симметричного ключа. Если поддерживается несколько алгоритмов симметричного шифрования, то симметричных ключей может быть несколько, по одному ключу каждого алгоритма. Все ключи хранятся в защищенном виде.

Пары ключей и симметричные ключи могут находиться только в контейнере. Только открытый ключ пары может находиться вне контейнера.

По умолчанию для каждого пользователя создается контейнер с именем этого пользователя. Можно создавать дополнительные контейнеры и назначать им произвольные имена, которые обязательно должны быть уникальными.

Рис. 2 – База данных криптографических ключей

Всякий раз, когда создается новый, используемый по умолчанию экземпляр класса RSACryptoServiceProvider, автоматически создается готовая к использованию новая пара открытого и личного ключей.

Допустимые размеры ключей в алгоритме RSA зависят от поставщика криптографических услуг (CSP), который используется экземпляром RSACryptoServiceProvider. Windows CSP разрешают размеры ключей от 384 до 16384 бит с шагом 8 бит для версий Windows до Windows 8.1 и размеры ключей от 512 до 16384 бит для Windows 8.1. Длина ключа по умолчанию при этом 1024 бита.

4. Сохранение ключей в контейнере ключей.

Следующий пример кода показывает, как создать новую пару ключей шифрования/расшифрования (сгенерировать новые параметры алгоритма RSA) и сохранить ключи в пользовательском контейнере ключей с заданным именем.

Пример взят из MSDN:

```
using System;
using System.Security.Cryptography;

class RSACSPSample
{
   static void Main()
   {
      string KeyContainerName = "MyKeyContainer";
      RSAPersistKeyInCSP(KeyContainerName);
      RSADeleteKeyInCSP(KeyContainerName);
   }
   public static void RSAPersistKeyInCSP(string ContainerName)
   {
      try
      {
         CspParameters cspParams = new CspParameters();
         cspParams.KeyContainerName = ContainerName;
         RSACryptoServiceProvider RSAalg = new RSACryptoServiceProvider(cspParams);
         Console.WriteLine("The RSA key was persisted in the container, \"{0}\".",
ContainerName);
      }
      catch(CryptographicException e)
      {
         Console.WriteLine(e.Message);
```

```
        }
      }
      public static void RSADeleteKeyInCSP(string ContainerName)
      {
        try
        {
          CspParameters cspParams = new CspParameters();
          cspParams.KeyContainerName = ContainerName;
          RSACryptoServiceProvider RSAalg = new RSACryptoServiceProvider(cspParams);
          RSAalg.PersistKeyInCsp = false;
          RSAalg.Clear();
          Console.WriteLine("The RSA key was deleted from the container, \"{0}\".",
ContainerName);
        }
        catch(CryptographicException e)
        {
          Console.WriteLine(e.Message);
        }
      }
    }
```

Если надо повторно использовать созданные ранее и сохраненные в пользовательском контейнере ключи, этого можно достичь, проинициализировав класс RSACryptoServiceProvider заполненным объектом CspParameters, используя соответствующий конструктор.

RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(cspParams);

5. Использования хранилища ключей компьютера.

Для использования хранилища ключей компьютера вместо хранилища ключей профилей пользователей нужно задать свойство UseMachineKeyStore класса RSACryptoServiceProvider:

RSACryptoServiceProvider. UseMachineKeyStore = true;

Все экземпляры класса RSACryptoServiceProvider, созданные с помощью конструктора по умолчанию (т.е. не инициализированные объектом CspParameters) будут использовать этот параметр.

Можно установить флаг использования хранилища ключей компьютера непосредственно для объекта класса CspParameters.

Операции криптопровайдера базируются на значении перечисления CspProviderFlags. В этом перечислении поддерживаются два значения: UseDefaultKeyContainer и UseMachineKeyStore. Если задан флаг UseDefaultKeyContainer, то информация о ключах будет считываться из контейнера ключей по умолчанию (т.е. из пользовательского контейнера). Если указан флаг UseMachineKeyStore, то информация о ключах будет считываться из контейнера ключей компьютера.

В следующем примере кода в начале создается экземпляр класса CspParameters, затем устанавливается флаг UseMachineKeyStore и задается имя контейнера ключей.

CspParameters cspParam = new CspParameters();

cspParam.Flags = CspProviderFlags. UseMachineKeyStore;

cspParam.KeyContainerName = “MachineKeyStore”;

6. Сохранение ключей в XML-формате.

Если надо повторно использовать созданные ранее ключи либо передать открытый ключ между двумя приложениями, использующими разные платформы или разные криптографические библиотеки, можно сохранить ключевую информацию в формате XML.

Рассмотрим фрагмент кода, в котором ключи шифрования/расшифрования сохраняются в двух XML-файлах. В файле PublicPrivateKey.xml сохраняется пара ключей: открытый и личный. В файле PublicOnlyKey.xml – только открытый ключ. Для сохранения ключей используется метод ToXmlString.

Сначала создаем экземпляр RSACryptoServiceProvider с ключами по умолчанию.

```
RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();
```

Затем сохраняем ключи в соответствующих файлах.

```
// Пару ключей
StreamWriter writer = new StreamWriter(“PublicPrivateKey.xml”);
string publicPrivateKeyXML = rsa. ToXmlString(true);
writer.Write(publicPrivateKeyXML);
writer.Close();
// Только открытый ключ
writer = new StreamWriter(“PublicOnlyKey.xml”);
string publicOnlyKeyXML = rsa. ToXmlString(false);
writer.Write(publicOnlyKeyXML);
writer.Close();
```

Теперь рассмотрим, как можно загрузить ключи из XML-файла с помощью метода FromXmlString().

```
RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();
StreamReader reader = new StreamReader(“PublicPrivateKey.xml”);
string publicPrivateKeyXML = reader.ReadToEnd();
rsa. FromXmlString(publicPrivateKeyXML);
reader.Close();
```

После извлечения ключей можно выполнять процедуру расшифрования данных.

Задания на лабораторную работу

Разработать программу, демонстрирующую применение алгоритма RSA для шифрования/расшифрования данных. Использовать класс RSACryptoServiceProvider.

Программа должна обеспечивать следующие функции:

1) Генерацию пары ключей шифрования/расшифрования и сохранение их в машинном контейнере ключей, пользовательском контейнере ключей с заданным именем, либо в XML-файлах в соответствии с вариантом задания.

2) Шифрование текстовой строки, введенной пользователем. Ключ шифрования должен извлекаться из контейнера ключей или XML-файла.

При шифровании отображать на экране:

а) используемые открытые параметры алгоритма RSA (модуль и показатель степени);

б) текущий размер ключа.

3) Расшифрование заданного шифртекста. Ключ расшифрования должен извлекаться из контейнера ключей или XML-файла. При расшифровании отображать на экране используемые параметры алгоритма RSA (P, Q, N, e, d).

4) Вывод информации о допустимых размерах ключей для используемой реализации алгоритма RSA.

Варианты заданий:

Размер ключей в битах / Хранение ключей	1024	2048	4096
в XML-файлах	1	2	3
в машинном контейнере ключей	4	5	6
в пользовательском контейнере ключей	7	8	9

Контрольные вопросы

1. Какие асимметричные алгоритмы поддерживает пространство имен System.Security.Cryptography?
2. Какова иерархия наследования криптографических классов асимметричных алгоритмов шифрования в .NET Framework?
3. Что такое CSP?
4. От чего зависят допустимые размеры ключей в алгоритме RSA? Каковы их значения?
5. Каков используемый по умолчанию размер ключей в алгоритме RSA? Можно ли задать размер ключей отличный от используемого по умолчанию? Как это сделать?
6. Как можно сохранить и использовать в дальнейшем сгенерированные один раз ключи для алгоритма RSA?
7. Что такое контейнер ключей? Какие бывают типы контейнеров ключей, для чего они используются, кто имеет к ним доступ? Какой тип контейнера ключей используется по умолчанию?